SWeat

파이썬 기초

복습 타임

조건문

지난 수업 복습

If – else 문

```
if 조건문:
    수행할 문장
else:
    수행할 문장
```

If – elif – else 문

```
if 조건문:
    수행할 문장
elif 조건문:
    수행할 문장
else:
    수행할 문장
```

반복문

지난 수업 복습

for loop

```
for [변수] in [리스트]: ( 또는 튜플, 문자열)
 [반복 실행할 구문]
 [반복 실행할 구문]
 [반복 실행할 구문]
          .
          .
          .
```

어느 구문까지 반복할지는 들여쓰기를 통해 구분

while loop

```
while [조건문]:
 [반복 실행할 구문]
 [반복 실행할 구문]
 [반복 실행할 구문]
          .
          .
          .
```

어느 구문까지 반복할지는 들여쓰기를 통해 구분

반복문

지난 수업 복습

for loop

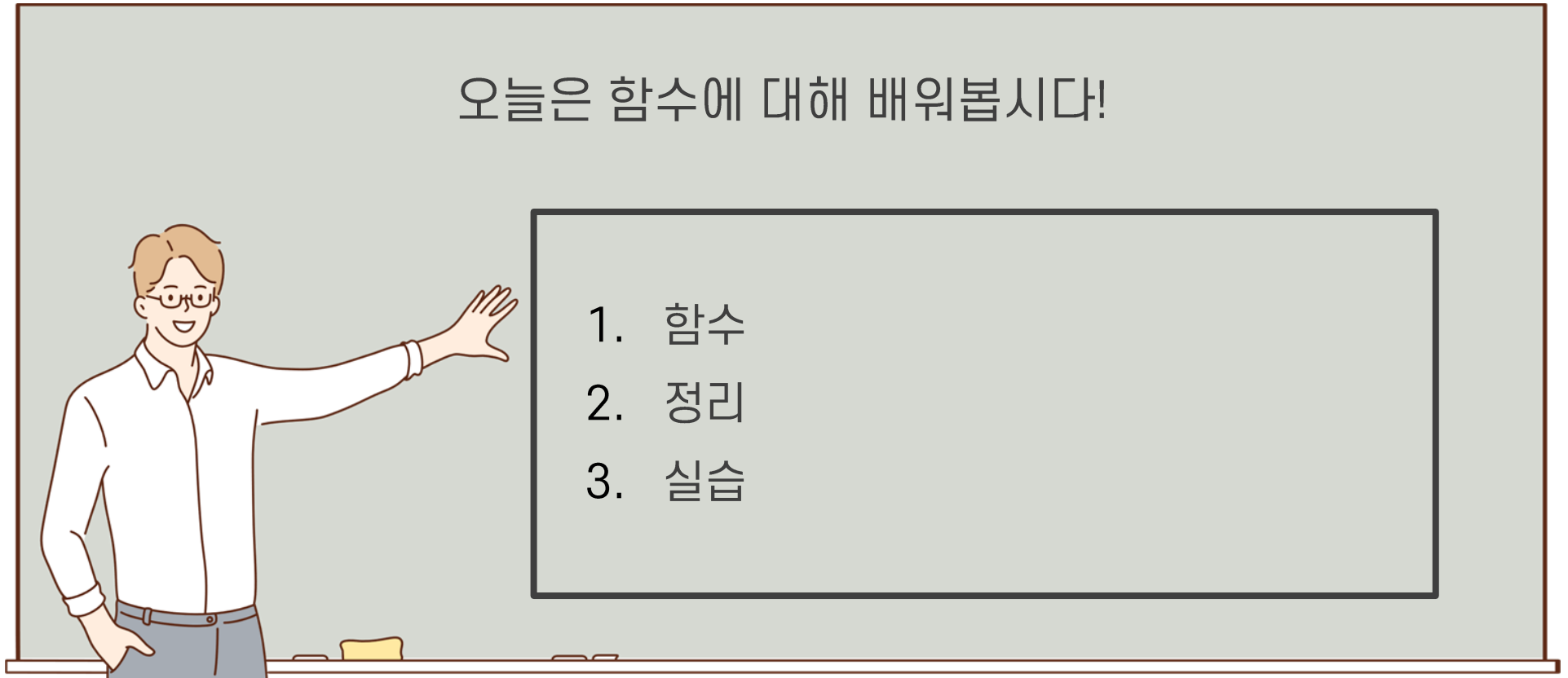
```python
for i in range(5) :
    print(i, end=" ")


for i in range(0,5) :
    print(i, end=" ")


for i in range(0, 5, 1) :
    print(i, end=" ")


for i in [0,1,2,3,4] :
    print(i, end=" ")
```

while loop

```python
i = 0
while True :
    print(i, end=" ")

    i = i + 1

    if (i >= 5) :
        break


i = 0
while(i < 5) :
    print(i, end=" ")

    i = i + 1
```

학습 목표

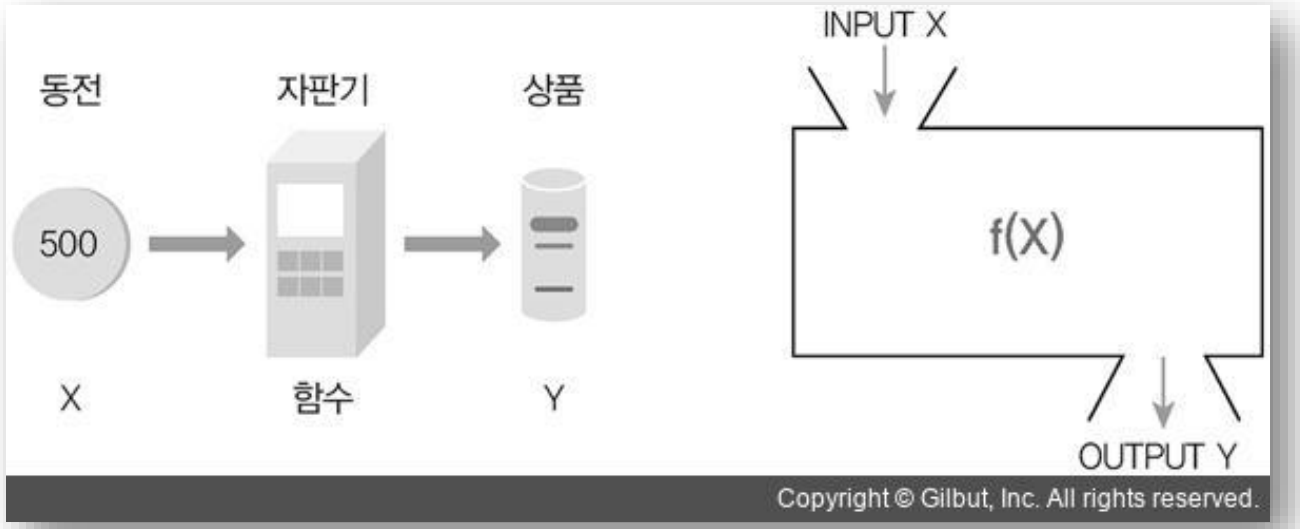
Learning Objectives



1. 함수

함수, 네가 궁금해

함수란?



함수 : 입력값(input)을 통해 특정 작업을 수행한 후, 그 결과물(output)을 내어 놓는 동작

함수를 사용하는 이유는 무엇일까?

함수란?

함수는 간편해요

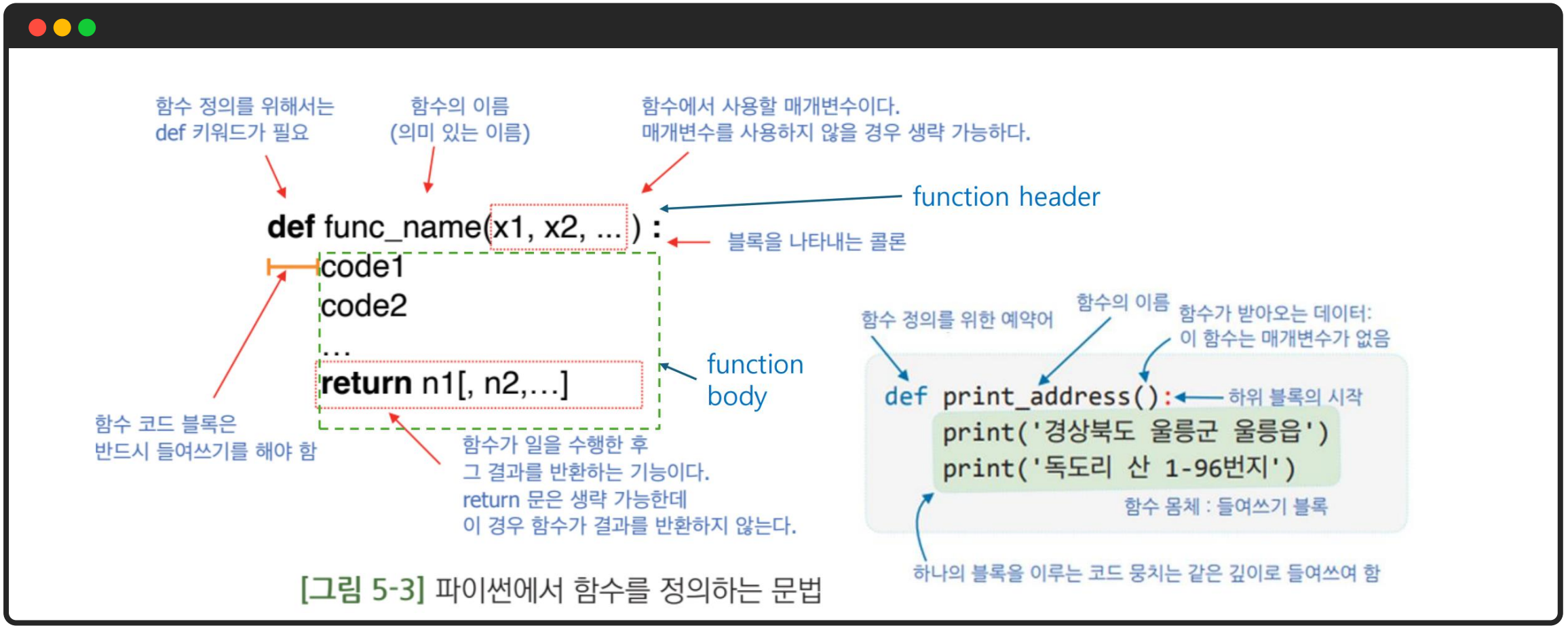
```
print("Hello, My name is Kim Sweat. Nice to meet you!")
print("Hello, My name is Kim Sweat. Nice to meet you!")
```

똑같은 코드를 반복적으로 작성하면,
코드가 길어질 때 가독성이 나빠지며,
매번 입력해야 하기 때문에 작업 효율성이 떨어져요

```
1   def hihi():
2       print("Hello, My name is Kim Sweat. Nice to meet you!")
3
4   hihi()
5   hihi()
```

함수의 구조

함수의 이해



[그림 5-3] 파이썬에서 함수를 정의하는 문법

함수의 구조

함수의 이해

함수의 구조는 다음과 같습니다.

```
def 함수이름(매개변수):
    <수행할 문장>
    ...
    return 결과값
```

📌 함수 예시

```
def add(a,b):
    sum = a+b
    return sum
```

이 add 함수는 두 가지의 변수를 입력 받고,
두 값을 더한 값을 return 합니다.

함수의 구조

함수의 이해

함수는 여러가지 구조가 있습니다.

📌 입력값(매개변수)과 출력값(반환값)이 없는 함수도 존재합니다.

```
1   def hihi():
2       print("Hello, My name is Kim Sweat. Nice to meet you!")
3
4   hihi()
5   hihi()
```

📌 입력값은 존재하지만 출력값은 없는, 반대로 입력값은 없지만 출력값은 있는 함수도 존재합니다.

```
def add(a,b):
    print("%d와 %d를 더한 값은 %d 입니다" %(a, b, a+b))
```

함수 호출

함수의 이해

함수를 만들었으면, 필요할 때 사용해야겠죠?
우리는 이러한 사용을 함수의 호출이라고 부릅니다.

```
def add(a,b):
    return a+b

a = 2
b = 3
sum = add(a,b)
print(sum);
```

함수는 어떻게 사용할까?

함수의 이해

📌 입력값, 리턴값이 모두 있을 때

```
def add(a,b):
    return a+b

a = 2
b = 3
sum = add(a,b)
print(sum);
```

📌 입력값 or 리턴값 둘 중 하나만 있을 때

```
def add(a,b):
    print("%d+%d = %d 입니다" %(a, b, a+b))

add(2,3)
```

```
def hihi():
    return "Hello!Hello!"

a = hihi()
print(a)
```

함수는 어떻게 사용할까?

함수의 이해

📌 리스트 활용

```
def list_fun():
    return [1,2,3]
print(list_fun())
```

📌 튜플 활용

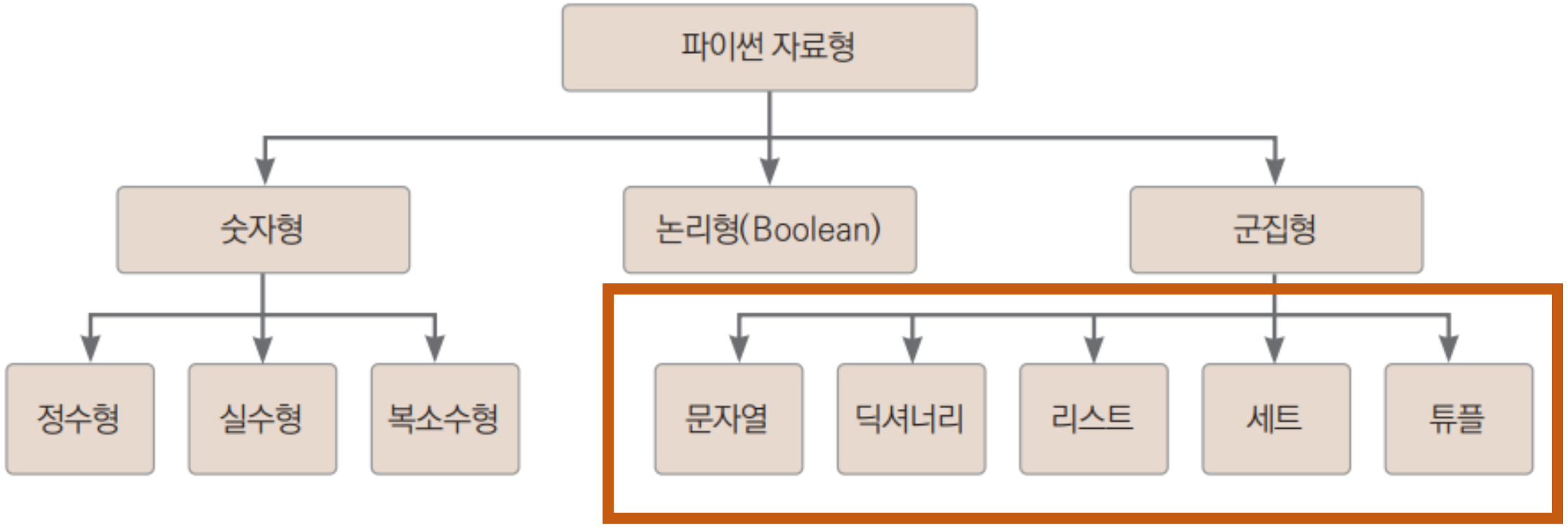
```
def plus_minus(a,b):
    return a+b, a-b
x , y = plus_minus(5,3)
z = plus_minus(6,3)
print(x,y)
print(z)
```

2. iterable 자료형

iterable 자료형은 어떤 것이 있을까?

iterable 자료형의 이해

▼ 파이썬 자료형



▼ 군집형

연산자	설명	순서	수정	생성자	예문
리스트	순서가 있는 데이터의 모음으로, 정의된 리스트라도 후에 삭제, 추가 등 변경할 수 있습니다.	○	○	[ ] or list()	[1, 2, 3]
튜플	리스트와 마찬가지로 순서가 있는 데이터이나, 한 번 정의되면 수정이 불가능합니다.	○	×	tuple()	(4, 5, 6)
딕셔너리	키와 값으로 구성된 자료구조로, 하나의 키에 여러 가지 값들을 매핑할 수도 있습니다. 또한 딕셔너리에서는 키를 사용해 그에 해당하는 값들을 쉽게 불러올 수 있습니다.	×	○	{ } or dict()	{'name': 'Sam', 'age': 20}
세트	중복된 값을 보유하지 않는 자료구조로, 리스트나 튜플과는 달리 순서가 없는 형태입니다.	×	○	set()	{1, 3, 5}
문자열	따옴표 안에 들어 있는 자료 형태를 말하며, 문자형이기 때문에 숫자형 같은 사칙연산은 불가능합니다. 숫자 데이터라도 '10'과 같이 따옴표 안에 들어 있으면 숫자형이 아닌 문자형으로 인식됩니다.	○	×	' ' or str()	'string'

리스트

iterable 자료형의 이해

▼ 군집형

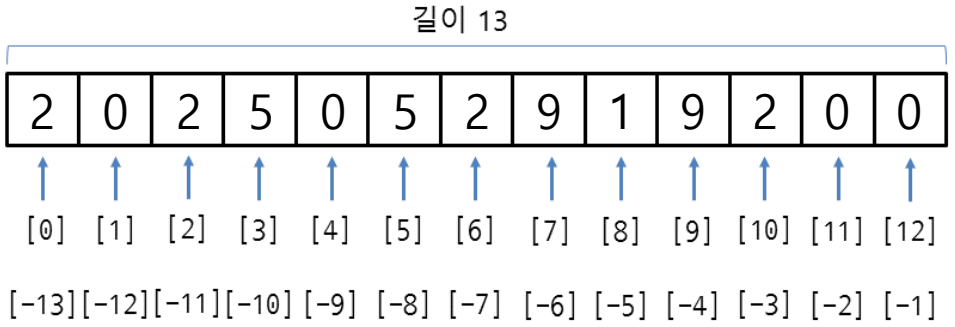
연산자	설명	순서	수정	생성자	예문
리스트	순서가 있는 데이터의 모음으로, 정의된 리스트라도 후에 삭제, 추가 등 변경할 수 있습니다.	○	○	[ ] or list()	[1, 2, 3]

리스트 형식

- 대괄호 사용 []

```
# 변수 = [value1,value2,value3,...,valueN]
```

```
listData = [2,0,2,5,0,5,2,9,1,9,2,0,0]
```



리스트

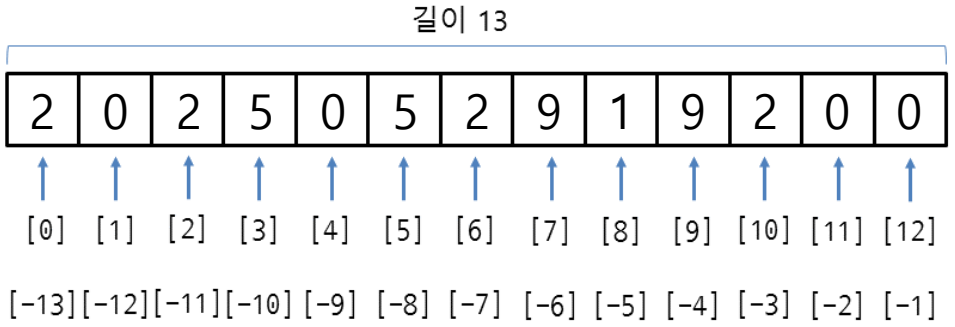
iterable 자료형의 이해

▼ 군집형

연산자	설명	순서	수정	생성자	예문
리스트	순서가 있는 데이터의 모음으로, 정의된 리스트라도 후에 삭제, 추가 등 변경할 수 있습니다.	○	○	[ ] or list()	[1, 2, 3]

리스트 생성

```
# 빈 리스트 생성
listData = list()   # 방법 1
listData = []       # 방법 2

# 리스트 생성과 동시에 값 할당
# 변수 = [value1,value2,value3,...,valueN]
listData = [2,0,2,5,0,5,2,9,1,9,2,0,0]
```



리스트

iterable 자료형의 이해

▼ 군집형

연산자	설명	순서	수정	생성자	예문
리스트	순서가 있는 데이터의 모음으로, 정의된 리스트라도 후에 삭제, 추가 등 변경할 수 있습니다.	○	○	[ ] or list()	[1, 2, 3]

특징

1. 순서 존재
2. 수정 가능
3. 중복 가능

요소 접근 방법

Index로 접근

```
listData = [0,0,0,0,1,1,1,1]
print(listData)
listData[2] = 4
print(listData)
```

✓ 0.0s

```
[0, 0, 0, 0, 1, 1, 1, 1]
[0, 0, 4, 0, 1, 1, 1, 1]
```

리스트

iterable 자료형의 이해

메소드	하는 일
index(x)	원소 x를 이용하여 위치를 찾는다. 원소 x의 인덱스 값을 반환한다.
append(x)	원소 x를 리스트의 끝에 추가한다.
count(x)	리스트 내에서 x 원소의 개수를 반환한다.
extend([x1, x2])	[x1, x2] 리스트를 기존 리스트에 삽입한다.
insert(index, x)	원하는 index 위치에 x를 추가한다.
remove(x)	x 원소를 리스트에서 삭제한다.
pop(index)	index 위치의 원소를 삭제한 후 반환한다. 이때 index는 생략될 수 있으며 이 경우 리스트의 마지막 원소를 삭제하고 이를 반환한다.
sort()	값을 오름차순 순서대로 정렬한다. 키워드 인자 reverse=True이면 내림차순으로 정렬한다.
reverse()	리스트를 원래 원소들의 역순으로 만들어 준다.

튜플

iterable 자료형의 이해

▼ 군집형

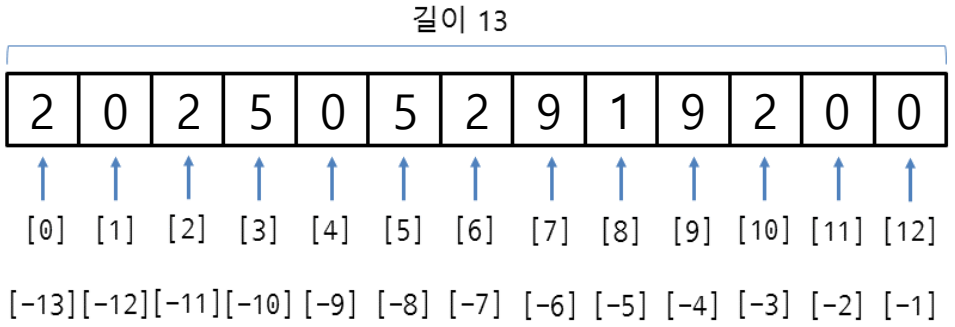
연산자	설명	순서	수정	생성자	예문
튜플	리스트와 마찬가지로 순서가 있는 데이터이나, 한 번 정의되면 수정이 불가능합니다.	○	×	( ) or tuple()	(4, 5, 6)

튜플 형식

- 소괄호 사용 ()

```
# 변수 = (value1,value2,value3,...,valueN)
```

```
tupleData = (2,0,2,5,0,5,2,9,1,9,2,0,0)
```



튜플

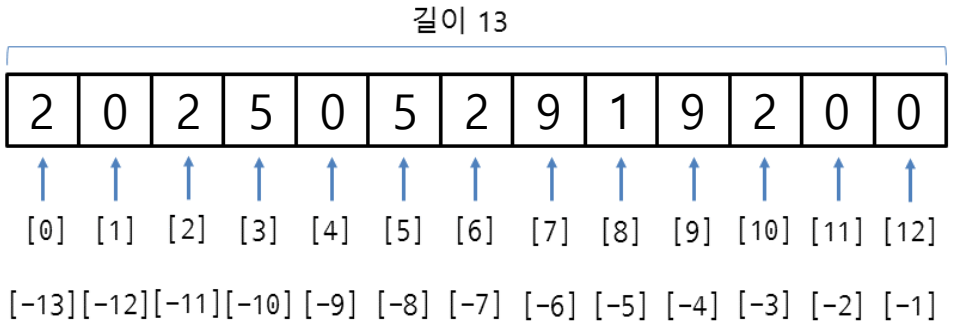
iterable 자료형의 이해

▼ 군집형

연산자	설명	순서	수정	생성자	예문
튜플	리스트와 마찬가지로 순서가 있는 데이터이나, 한 번 정의되면 수정이 불가능합니다.	○	×	( ) or tuple()	(4, 5, 6)

튜플 생성

```
# 빈 튜플 생성(무의미)
tupleData = tuple()

# 튜플 생성과 동시에 값 할당
# 변수 = (value1,value2,value3,...,valueN)
tupleData = (2,0,2,5,0,5,2,9,1,9,2,0,0)
```



튜플

iterable 자료형의 이해

▼ 군집형

연산자	설명	순서	수정	생성자	예문
튜플	리스트와 마찬가지로 순서가 있는 데이터이나, 한 번 정의되면 수정이 불가능합니다.	○	×	( ) or tuple()	(4, 5, 6)

특징

1. 순서 존재
2. 수정 불가능
3. 중복 가능

요소 접근 방법

Index로 접근

```
tupleData = (0,0,0,0,1,1,1,1)
print(tupleData)
tupleData[2] = 4
print(tupleData)
```

⊗ 0.0s

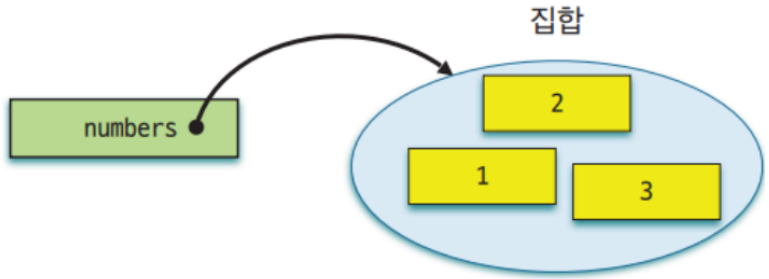
```
(0, 0, 0, 0, 1, 1, 1, 1)

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[41], line 10
      8 tupleData = (0,0,0,0,1,1,1,1)
      9 print(tupleData)
---> 10 tupleData[2] = 4
     11 print(tupleData)

TypeError: 'tuple' object does not support item assignment
```

집합(set)

iterable 자료형의 이해

▼ 군집형

연산자	설명	순서	수정	생성자	예문
세트	중복된 값을 보유하지 않는 자료구조로, 리스트나 튜플과는 달리 순서가 없는 형태입니다.	X	○	{ } or set()	{1, 3, 5}



집합 형식

- 중괄호 사용 {}

```
# 변수 = {value1,value2,value3,...,valueN}
```

```
setData = {2,0,2,5,0,5,2,9,1,9,2,0}
print(setData)
```

✓ 0.0s

```
{0, 1, 2, 5, 9}
```

* 중복값은 삭제

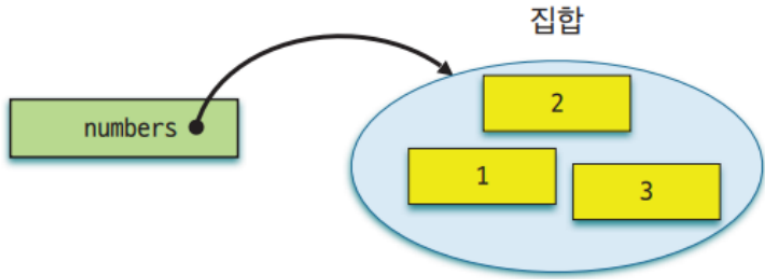
* 순서는 자동으로 오름차순 정렬

집합(set)

iterable 자료형의 이해

▼ 군집형

연산자	설명	순서	수정	생성자	예문
세트	중복된 값을 보유하지 않는 자료구조로, 리스트나 튜플과는 달리 순서가 없는 형태입니다.	X	○	{ } or set()	{1, 3, 5}



집합 생성

```
# 빈 집합 생성
setData = set()

# 집합 생성과 동시에 값 할당
# 변수 = {value1,value2,value3,...,valueN}
setData = {2,0,2,5,0,5,2,9,1,9,2,0}
```

집합(set)

iterable 자료형의 이해

▼ 군집형

연산자	설명	순서	수정	생성자	예문
세트	중복된 값을 보유하지 않는 자료구조로, 리스트나 튜플과는 달리 순서가 없는 형태입니다.	X	○	{ } or set()	{1, 3, 5}

특징

1. 순서 미존재
2. 수정 가능
3. 중복 불가능

요소 접근 방법

Index로 접근 불가

```
setData = {0,0,0,0,1,1,1,1}
print(setData)
setData[2] = 4
print(setData)
```

0.0s

```
{0, 1}

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[45], line 10
      8 setData = {0,0,0,0,1,1,1,1}
      9 print(setData)
---> 10 setData[2] = 4
     11 print(setData)

TypeError: 'set' object does not support item assignment
```

* 중복 값은 삭제
* 순서는 자동으로 오름차순 정렬

집합(set)

iterable 자료형의 이해

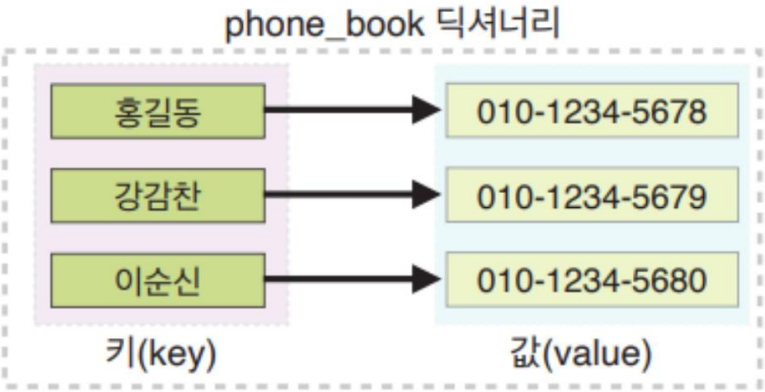
메소드	하는 일
add(x)	집합 내의 특정 x 원소를 추가한다.
discard(x)	집합 내의 특정 x 원소를 삭제한다.
clear()	집합 내의 모든 원소를 삭제한다.
union(s)	s 집합과의 합집합을 구한다. | 연산과 동일하다.
difference(s)	s 집합과의 차집합을 구한다. - 연산과 동일하다.
intersection(s)	s 집합과의 교집합을 구한다. & 연산과 동일하다.
symmetric_difference(s)	s 집합과의 대칭 차집합을 구한다. ^ 연산과 동일하다.
issubset(s)	s 집합의 부분집합인가를 구한다. True/False를 반환한다.
issuperset(s)	; 집합의 상위집합인가를 구한다. True/False를 반환한다.
isdisjoint(s)	; 집합과 서로소인가를 구한다. True/False를 반환한다.

딕셔너리

iterable 자료형의 이해

▼ 군집형

연산자	설명	순서	수정	생성자	예문
딕셔너리	키와 값으로 구성된 자료구조로, 하나의 키에 여러 가지 값들을 매핑할 수도 있습니다. 또한 딕셔너리에서는 키를 사용해 그에 해당하는 값들을 쉽게 불러올 수 있습니다.	×	○	{ } or dict()	{'name': 'Sam', 'age': 20}



딕셔너리 형식

- 중괄호 사용 {} * 집합과 동일하지만 요소 저장 방식 차이남

```
# 변수 = {key1:value1,key2:value2,...,key3:valueN}
```

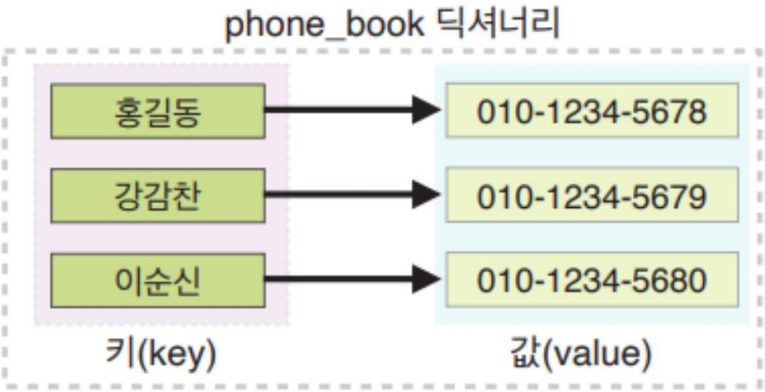
```
dictData = {"year":2025,"month":5,"day":29,"hour":19,"min":20,"sec":0}
```

딕셔너리

iterable 자료형의 이해

▼ 군집형

연산자	설명	순서	수정	생성자	예문
딕셔너리	키와 값으로 구성된 자료구조로, 하나의 키에 여러 가지 값들을 매핑할 수도 있습니다. 또한 딕셔너리에서는 키를 사용해 그에 해당하는 값들을 쉽게 불러올 수 있습니다.	×	○	{ } or dict()	{'name': 'Sam', 'age': 20}



집합 생성

```
# 빈 딕셔너리 생성
dictData = dict()

# 딕셔너리 생성과 동시에 값 할당
# 변수 = {key1:value1,key2:value2,...,key3:valueN}
dictData = {"year":2025,"month":5,"day":29,"hour":19,"min":20,"sec":0}
```

딕셔너리

iterable 자료형의 이해

▼ 군집형

연산자	설명	순서	수정	생성자	예문
딕셔너리	키와 값으로 구성된 자료구조로, 하나의 키에 여러 가지 값들을 매핑할 수도 있습니다. 또한 딕셔너리에서는 키를 사용해 그에 해당하는 값들을 쉽게 불러올 수 있습니다.	×	○	{ } or dict()	{'name': 'Sam', 'age': 20}

특징

1. 순서 미존재
2. 수정 가능
3. Key 중복 불가능
3. Value 중복 가능

요소 접근 방법

Key로 접근

```
dictData = {"year":2025,"month":5,"day":29,"hour":19,"min":20,"sec":0}
print(dictData)
dictData["year"] = 2035
print(dictData)
```

✓ 0.0s

```
{'year': 2025, 'month': 5, 'day': 29, 'hour': 19, 'min': 20, 'sec': 0}
{'year': 2035, 'month': 5, 'day': 29, 'hour': 19, 'min': 20, 'sec': 0}
```

딕셔너리

iterable 자료형의 이해

메소드	하는 일
keys()	딕셔너리 내의 모든 키를 반환한다.
values()	딕셔너리 내의 모든 값을 반환한다.
items()	딕셔너리 내의 모든 항목을 [키]:[값] 쌍으로 반환한다.
get(key)	키에 대한 값을 반환한다. 키가 없으면 None을 반환한다.
pop(key)	키에 대한 값을 반환하고, 그 항목을 삭제한다. 키가 없으면 KeyError 예외를 발생시킨다.
popitem()	제일 마지막에 입력된항목을 반환하고 그 항목을 삭제한다.
clear()	딕셔너리 내의 모든 항목을 삭제한다.

오늘 배운 내용에서 질문 있나요?